Міністерство освіти і науки України

Національний університет „Львівська політехніка”

Кафедра ЕОМ



**Звіт**

**з ЛабораторнОЇ роботИ №6**

З дисципліни: “Комп’ютерні системи”

Тема «Побудова функціональної схеми процесорів архітектури RISC CPU»

Виконав: ст. гр. КІ-33

Харченко М.В.

Прийняв: викладач каф. ЕОМ

Козак Н.Б.

**Львів 2020**

**Мета:** навчитись розробляти нові функціональні модулі з метою розширення функціонування базової архітектури процесорів RISC, що відповідає програмній моделі, а також створення функціональної схеми.

**Завдання до лабораторної роботи**

Згідно варіанту в журналі та узгоджено з викладачем обрати один з процесорів заданого сімейства на базі RISC архітектури, в звіті навести основну інформацію по процесору, схему та характеристику.

Внести необхідні зміни в проект з 5-ї лабораторної (типи кешів, розмірності шин даних, назви та типи сигналів, операції, буферні модулі і т.д.) з метою реалізації фукнціоналу з обраного процесору. В звіті представити коротко внесені зміни з описом ходу роботи, а також кінцеву функціональну схему програмної моделі.

**Порядок виконання роботи**

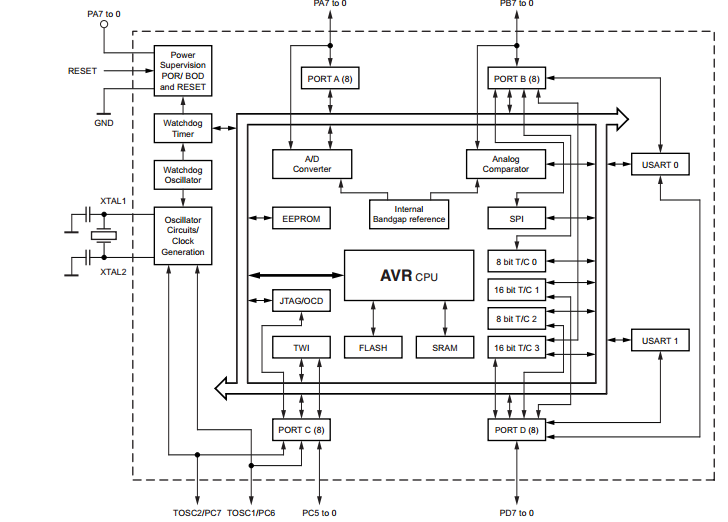
1. Реалізувати поставлене завдання до лабораторної роботи;
2. Для кожного з блоків, що входять до складу розробленої програмної моделі на базі RISC CPU визначити вхідні і вихідні сигнали інтерфейсів;
3. Побудувати повну функціональну схему розробленої програмної моделі на базі RISC CPU;
4. Розглянути взаємозв`язок блоків у структурі і визначити можливі типи конвеєрів.

**Варіант 4**

**Теоретична частина**

**Основні характеристики мікроконтролера Atmel AVR ATmega**

ATMEL AVR ATmega - це CMOS 8-ми бітний мікроконтролер, який базується на AVR® RISK архітектурі. Виконуючи потужні інструкції за один такт, мікроконтролер ATmega164P-B/324P-B/644P-B досягає продуктивності 1 MIPS, що дозволяє розробнику системи оптимізувати енергоспоживання залежно від швидкості обробки.

 **Загальна схема мікроконтролера Atmel AVR ATmega**

AVR ядро поєднує в собі багатий набір інструкцій з 32-ма регістрами загального призначення. Всі 32 регістри безпосередньо підключені до ALU, дозволяючи доступ двом незалежним регістрам в одній інструкції, за один такт. Отримана архітектура є більш ефективною, досягається пропускна здатність до десяти разів швидше, ніж у звичайних CISC мікроконтролерів.

**Хід роботии**

- 32 х 8 регістрів загального призначення.

- Двадцять три вектора переривань .

- Підтримка до 128 Кб програми і до 64 Кбайт пам'яті даних.

• Периферійні функції:

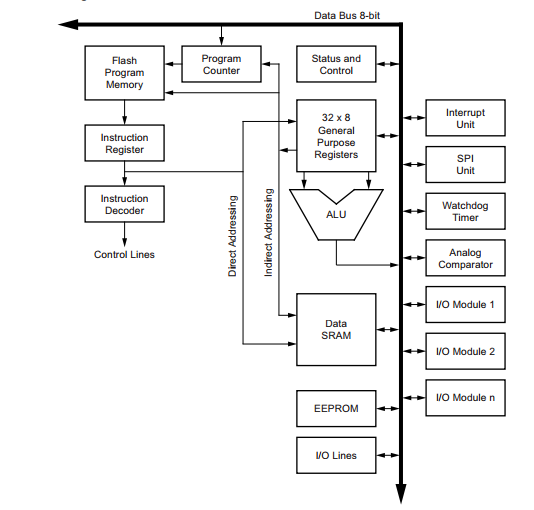
- Програмований UART.

- Два 8-розрядний таймер / лічильники.

- Вісім зовнішніх джерел переривань

- Два паралельних порти.

**Блок-схема процесора:**



Для того, щоб максимально збільшити продуктивність і паралелізм, використовується Гарвардська архітектуру. Інструкції в пам'яті програми виконуються за допомогою одного конвеєризованого рівня. У той час як одна команда виконується, наступна команда попередньо витягується з пам'яті програми. Пам'ять програми в системі - перепрограмуюча флеш-пам'ять. Регістровий файл швидкого доступу містить 32 х 8-бітних регістри загального призначення з одним часом доступу тактового циклу. У типовій операції АЛУ, два операнда виводяться з реєстрового файлу, виконується операція, і результат зберігається в файлі регістрів - за один такт.

**Код програми:**

**.reg\_file.h**

#ifndef \_REG\_FILE\_H

#define \_REG\_FILE\_H

#include <systemc.h>

#include "config.h"

#include "constants.h"

SC\_MODULE(reg\_file)

{

sc\_in<bool> clk;

sc\_in<bool> nrst;

sc\_in<sc\_lv<8> > reg\_rd\_in;

sc\_out<sc\_lv<8> > reg\_rd\_out;

sc\_in<sc\_lv<5> > reg\_rd\_adr;

sc\_out<sc\_lv<8> > reg\_rr\_out;

sc\_in<sc\_lv<5> > reg\_rr\_adr;

sc\_in<sc\_logic> reg\_rd\_wr;

sc\_in<sc\_logic> post\_inc;

sc\_in<sc\_logic> pre\_dec;

sc\_in<sc\_logic> reg\_h\_wr;

sc\_out<sc\_lv<16> > reg\_h\_out;

sc\_in<sc\_lv<3> > reg\_h\_adr;

sc\_out<sc\_lv<16> > reg\_z\_out;

sc\_in<sc\_uint<5> > regadr;

sc\_in<sc\_logic> regre;

sc\_in<sc\_logic> regwe;

sc\_out<sc\_lv<8> > reg\_in; // NB! Reverse direction in naming!

sc\_in<sc\_lv<8> > reg\_out;

sc\_signal<sc\_lv<8> > r26h;

sc\_signal<sc\_lv<8> > r27h;

sc\_signal<sc\_lv<8> > r28h;

sc\_signal<sc\_lv<8> > r29h;

sc\_signal<sc\_lv<8> > r30h;

sc\_signal<sc\_lv<8> > r31h;

sc\_signal<sc\_lv<16> > sg\_adr16\_postinc;

sc\_signal<sc\_lv<16> > sg\_adr16\_predec;

sc\_signal<sc\_lv<16> > reg\_h\_in;

sc\_signal<sc\_lv<16> > sg\_tmp\_h\_data;

sc\_signal<sc\_lv<8> > register\_file[32];

void register\_write();

void r26to31();

void regz\_output();

void output\_regs();

void h\_dat\_mux();

void addr\_bus\_output();

SC\_CTOR(reg\_file)

{

SC\_METHOD(register\_write);

sensitive\_pos << clk;

SC\_METHOD(r26to31);

sensitive\_pos << clk;

SC\_METHOD(regz\_output);

sensitive << r31h << r30h;

SC\_METHOD(output\_regs);

sensitive << register\_file[0] << register\_file[1] << register\_file[2] << register\_file[3] << register\_file[4] << register\_file[5] << register\_file[6] << register\_file[7] << register\_file[8] << register\_file[9] << register\_file[10] << register\_file[11] << register\_file[12] << register\_file[13] << register\_file[14] << register\_file[15] << register\_file[16] << register\_file[17] << register\_file[18] << register\_file[19] << register\_file[20] << register\_file[21] << register\_file[22] << register\_file[23] << register\_file[24] << register\_file[25];

sensitive << reg\_rd\_adr << reg\_rr\_adr;

sensitive << r26h << r27h << r28h << r29h << r30h << r31h;

sensitive << regre << regadr;

SC\_METHOD(h\_dat\_mux);

sensitive << r26h << r27h << r28h << r29h << r30h << r31h << reg\_h\_adr << sg\_tmp\_h\_data;

SC\_METHOD(addr\_bus\_output);

sensitive << sg\_tmp\_h\_data << post\_inc << pre\_dec << sg\_adr16\_predec;

};

};

#endif

**.simple\_timer.h**

#ifndef \_SIMPLE\_TIMER\_H

#define \_SIMPLE\_TIMER\_H

#include <systemc.h>

#include "config.h"

#include "constants.h"

SC\_MODULE(simple\_timer)

{

sc\_in<bool> cp2;

sc\_in<bool> ireset;

sc\_out<sc\_logic> irqline;

sc\_in<sc\_logic> timer\_irqack;

sc\_signal<sc\_lv<8> > counter;

sc\_signal<sc\_logic> irqline\_int;

sc\_signal<sc\_logic> count\_end;

void main\_counter();

void irq\_out();

void set\_signals();

SC\_CTOR(simple\_timer)

{

SC\_METHOD(main\_counter);

sensitive\_pos << cp2;

SC\_METHOD(irq\_out);

sensitive\_pos << cp2;

SC\_METHOD(set\_signals);

sensitive << counter << irqline\_int;

}

};

**.ALU\_AVR.h**

#ifndef \_ALU\_AVR\_H

#define \_ALU\_AVR\_H

#include <systemc.h>

#include "config.h"

#define ADD\_SUMMATOR(x) v\_adder\_out[x] = v\_adder\_d\_in[x] ^ v\_adder\_r\_in[x] ^ v\_adder\_carry[x-1]; \

v\_adder\_carry[x] = ((v\_adder\_d\_in[x] ^ v\_adder\_nadd\_sub) & v\_adder\_r\_in[x]) | \

(((v\_adder\_d\_in[x] ^ v\_adder\_nadd\_sub) | v\_adder\_r\_in[x]) & v\_adder\_carry[x-1]);

#define NEG\_SUMMATOR(x) v\_neg\_op\_out[x] = ~v\_alu\_data\_d\_in[x] ^ v\_neg\_op\_carry[x-1]; \

v\_neg\_op\_carry[x] = ~v\_alu\_data\_d\_in[x] & v\_neg\_op\_carry[x-1];

#define INCDEC\_GENERATE(x) v\_incdec\_op\_out[x] = v\_alu\_data\_d\_in[x] ^ v\_incdec\_op\_carry[x-1]; \

v\_incdec\_op\_carry[x] = (v\_alu\_data\_d\_in[x] ^ idc\_dec) & v\_incdec\_op\_carry[x-1];

#define ALUMUX\_GENERATE(x) v\_alu\_data\_out\_int[x] = (v\_adder\_out[x] & (idc\_add | idc\_adc | idc\_adiw | adiw\_st | idc\_sub | idc\_subi | idc\_sbc | idc\_sbci | idc\_sbiw | sbiw\_st | idc\_cpse | idc\_cp | idc\_cpc | idc\_cpi)) | \

(v\_neg\_op\_out[x] & idc\_neg) | \

(v\_incdec\_op\_out[x] & (idc\_inc | idc\_dec)) | \

(v\_com\_op\_out[x] & idc\_com) | \

(v\_and\_op\_out[x] & (idc\_and | idc\_andi)) | \

(v\_or\_op\_out[x] & (idc\_or | idc\_ori)) | \

(v\_eor\_op\_out[x] & idc\_eor) | \

(v\_right\_shift\_out[x] & (idc\_lsr | idc\_ror | idc\_asr)) | \

(v\_swap\_out[x] & idc\_swap);

SC\_MODULE(alu\_avr)

{

sc\_in<sc\_lv<8> > alu\_data\_r\_in;

sc\_in<sc\_lv<8> > alu\_data\_d\_in;

sc\_in<sc\_logic> alu\_c\_flag\_in;

sc\_in<sc\_logic> alu\_z\_flag\_in;

// Operation signal inputs

sc\_in<sc\_logic> idc\_add;

sc\_in<sc\_logic> idc\_adc;

sc\_in<sc\_logic> idc\_adiw;

sc\_in<sc\_logic> idc\_sub;

sc\_in<sc\_logic> idc\_subi;

sc\_in<sc\_logic> idc\_sbc;

sc\_in<sc\_logic> idc\_sbci;

sc\_in<sc\_logic> idc\_sbiw;

sc\_in<sc\_logic> adiw\_st;

sc\_in<sc\_logic> sbiw\_st;

sc\_in<sc\_logic> idc\_and;

sc\_in<sc\_logic> idc\_andi;

sc\_in<sc\_logic> idc\_or;

sc\_in<sc\_logic> idc\_ori;

sc\_in<sc\_logic> idc\_eor;

sc\_in<sc\_logic> idc\_com;

sc\_in<sc\_logic> idc\_neg;

sc\_in<sc\_logic> idc\_inc;

sc\_in<sc\_logic> idc\_dec;

sc\_in<sc\_logic> idc\_cp;

sc\_in<sc\_logic> idc\_cpc;

sc\_in<sc\_logic> idc\_cpi;

sc\_in<sc\_logic> idc\_cpse;

sc\_in<sc\_logic> idc\_lsr;

sc\_in<sc\_logic> idc\_ror;

sc\_in<sc\_logic> idc\_asr;

sc\_in<sc\_logic> idc\_swap;

// Data output

sc\_out<sc\_lv<8> > alu\_data\_out;

// Flags output

sc\_out<sc\_logic> alu\_c\_flag\_out;

sc\_out<sc\_logic> alu\_z\_flag\_out;

sc\_out<sc\_logic> alu\_n\_flag\_out;

sc\_out<sc\_logic> alu\_v\_flag\_out;

sc\_out<sc\_logic> alu\_s\_flag\_out;

sc\_out<sc\_logic> alu\_h\_flag\_out;

//

// Internal signals

//

// \*int signals are inputs to a mux

sc\_signal<sc\_lv<8> > alu\_data\_out\_int;

// ALU flags

sc\_signal<sc\_logic> alu\_z\_flag\_out\_int;

sc\_signal<sc\_logic> alu\_c\_flag\_in\_int;

sc\_signal<sc\_logic> alu\_n\_flag\_out\_int;

sc\_signal<sc\_logic> alu\_v\_flag\_out\_int;

sc\_signal<sc\_logic> alu\_c\_flag\_out\_int;

// Adder signals

sc\_signal<sc\_logic> adder\_nadd\_sub;

sc\_signal<sc\_logic> adder\_v\_flag\_out;

sc\_signal<sc\_lv<9> > adder\_carry;

sc\_signal<sc\_lv<9> > adder\_d\_in;

sc\_signal<sc\_lv<9> > adder\_r\_in;

sc\_signal<sc\_lv<9> > adder\_out;

// Neg operator signals

sc\_signal<sc\_lv<8> > neg\_op\_in;

sc\_signal<sc\_lv<9> > neg\_op\_carry;

sc\_signal<sc\_lv<9> > neg\_op\_out;

// Inc, dec operator signals

sc\_signal<sc\_lv<8> > incdec\_op\_in;

sc\_signal<sc\_lv<8> > incdec\_op\_carry;

sc\_signal<sc\_lv<8> > incdec\_op\_out;

// Misc. signals

sc\_signal<sc\_lv<8> > com\_op\_out;

sc\_signal<sc\_lv<8> > and\_op\_out;

sc\_signal<sc\_lv<8> > or\_op\_out;

sc\_signal<sc\_lv<8> > eor\_op\_out;

// Shift signals

sc\_signal<sc\_lv<8> > right\_shift\_out;

// Swap signals

sc\_signal<sc\_lv<8> > swap\_out;

void addsubcontrol();

void aluinputs();

void signalsout();

void neg\_operation();

void incdec\_operation();

void com\_operation();

void bool\_operation();

void shift\_operation();

void swap\_operation();

void set\_adder\_x\_in();

void output\_mux();

SC\_CTOR(alu\_avr)

{

SC\_METHOD(addsubcontrol);

sensitive << idc\_add << idc\_adc << idc\_adiw << idc\_sub << idc\_subi;

sensitive << idc\_sbc << idc\_sbci << idc\_sbiw << adiw\_st << sbiw\_st;

sensitive << idc\_cp << idc\_cpc << idc\_cpi << idc\_cpse << idc\_ror;

sensitive << alu\_c\_flag\_in << alu\_z\_flag\_in << alu\_z\_flag\_out\_int;

SC\_METHOD(aluinputs);

sensitive << alu\_data\_d\_in << alu\_data\_r\_in << adder\_d\_in << adder\_r\_in << adder\_carry;

sensitive << alu\_c\_flag\_in\_int << adder\_nadd\_sub;

SC\_METHOD(signalsout);

sensitive << alu\_n\_flag\_out\_int << alu\_v\_flag\_out\_int << alu\_c\_flag\_out\_int << alu\_data\_out\_int;

SC\_METHOD(neg\_operation);

sensitive << alu\_data\_d\_in << neg\_op\_carry;

SC\_METHOD(incdec\_operation);

sensitive << alu\_data\_d\_in << incdec\_op\_carry << idc\_dec << incdec\_op\_out;

SC\_METHOD(com\_operation);

sensitive << alu\_data\_d\_in;

SC\_METHOD(bool\_operation);

sensitive << alu\_data\_d\_in << alu\_data\_r\_in;

SC\_METHOD(shift\_operation);

sensitive << idc\_ror << alu\_c\_flag\_in\_int << idc\_asr << alu\_data\_d\_in;

SC\_METHOD(swap\_operation);

sensitive << alu\_data\_d\_in;

SC\_METHOD(set\_adder\_x\_in);

sensitive << alu\_data\_d\_in << alu\_data\_r\_in;

SC\_METHOD(output\_mux);

sensitive << idc\_add << idc\_adc << idc\_adiw << idc\_sub << idc\_subi;

sensitive << idc\_sbc << idc\_sbci << idc\_sbiw << adiw\_st << sbiw\_st;

sensitive << idc\_and << idc\_andi << idc\_or << idc\_ori << idc\_eor << idc\_com << idc\_neg;

sensitive << idc\_inc << idc\_dec << idc\_cp << idc\_cpc << idc\_cpi << idc\_cpse << idc\_ror;

sensitive << idc\_lsr << idc\_ror << idc\_asr << idc\_swap;

sensitive << adder\_out << incdec\_op\_out << com\_op\_out << and\_op\_out << or\_op\_out << eor\_op\_out << right\_shift\_out << swap\_out;

sensitive << adder\_carry << alu\_z\_flag\_out\_int << alu\_c\_flag\_out\_int << incdec\_op\_carry;

sensitive << neg\_op\_carry << alu\_data\_d\_in << adder\_v\_flag\_out << alu\_v\_flag\_out\_int;

sensitive << alu\_data\_out\_int << neg\_op\_out << alu\_n\_flag\_out\_int;

}

};

#endif

#ifndef \_INSTMEMORY\_H

#define \_INSTMEMORY\_H

**.memory.h**

#include <systemc.h>

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include "config.h"

SC\_MODULE(instmemory)

{

sc\_in<bool> clk;

sc\_in<bool> reset;

sc\_in<sc\_uint<16> > pc;

sc\_out<sc\_lv<16> > inst;

sc\_lv<16> x[INSTMEMSIZE];

char \*memoryname;

// unsigned int pc;

void mread();

#ifdef \_CC\_MEMDUMP\_

int memcontents;

#endif

typedef instmemory SC\_CURRENT\_USER\_MODULE;

instmemory(sc\_module\_name name, char \*contents\_file)

{

unsigned int i = 0;

unsigned long line;

unsigned int codepos = 0;

unsigned char \*buf;

unsigned char \*code;

#ifdef \_CC\_MEMDUMP\_

unsigned int ci = 0x00000000;

// Create blank memory file filled with 0's

memcontents = open("mem.bin", O\_CREAT | O\_RDWR, 0666);

for (i = 0; i < (INSTMEMSIZE/4); i++)

write(memcontents, &ci, 4);

close(memcontents);

#endif

FILE \*fid = fopen(contents\_file, "r");

buf = (unsigned char\*)malloc(32768);

int size = fread(buf, 1, 32768, fid);

code = (unsigned char\*)malloc(INSTMEMSIZE);

memcpy(code,buf,size);

codepos = size;

i = 0;

while (i < codepos)

{

line = code[i];

line += (code[i+1]<<8);

#ifdef DEBUG\_MEMORY

printf("%04d : 0x%04X\n", i/2, line);

//fprintf(stdout, "%d : %X\n", i>>2, line);

#endif

x[i/2] = (sc\_lv<16>)line;

i += 2;

}

SC\_METHOD(mread);

sensitive\_neg << clk;

}

};

#endif

**Висновок:** на цій лабораторній роботі янавчився розробляти нові функціональні модулі з метою розширення функціонування базової архітектури процесорів RISC, що відповідає програмній моделі, а також створив функціональну схему.